An Attack on CFB Mode Encryption As Used By OpenPGP

Serge Mister & Robert Zuccherato

Entrust, Inc., 1000 Innovation Drive, Ottawa, Ontario, Canada K2K 3E7
{serge.mister,robert.zuccherato}@entrust.com

Abstract. This paper describes an adaptive-chosen-ciphertext attack on the Cipher Feedback (CFB) mode of encryption as used in OpenPGP. In most circumstances it will allow an attacker to determine 16 bits of any block of plaintext with about 2^{15} oracle queries for the initial setup work and 2^{15} oracle queries for each block. Standard CFB mode encryption does not appear to be affected by this attack. It applies to a particular variation of CFB used by OpenPGP. In particular it exploits an ad-hoc integrity check feature in OpenPGP which was meant as a "quick check" to determine the correctness of the decrypting symmetric key.

Keywords: OpenPGP, Cipher-Feedback Mode, encryption, chosen-ciphertext attacks

1 Introduction

The OpenPGP Message Format is described in RFC 2440 [4]. It is a very popular and commonly used format for signing and encrypting data files, particularly for signing and encrypting email. The formats described in the OpenPGP RFC have been implemented in a wide variety of popular freeware and commercial encryption products. Symmetric encryption in OpenPGP is performed using a variant of the standard Cipher-Feedback (CFB) Mode for block ciphers.

Adaptive chosen-ciphertext attacks on cryptographic protocols allow an attacker to decrypt a ciphertext C, getting the plaintext M, by submitting a series of chosen- ciphertexts $C' \neq C$ to an oracle which returns information on the decryption. The ciphertexts can be adaptively chosen so that information on previous decryptions is available before the next chosen ciphertext is submitted. These attacks have been used in the past to attack the RSA PKCS #1 v1.5 [11] encryption scheme [3], the Cipher-Block-Chaining (CBC) Mode of encryption when used with certain exploitable redundancies (e.g. padding schemes) [13, 2] and the OpenPGP CFB mode [12, 10] itself. The attack on the OpenPGP CFB mode was able to obtain the entire plaintext using one oracle query which returned to the attacker the entire decryption of C'.

This paper describes an adaptive chosen-ciphertext attack on the OpenPGP CFB mode of encryption. The attack requires an oracle that returns information on an ad-hoc integrity check in the OpenPGP CFB mode. We show that this oracle is likely instantiated in most applications that include OpenPGP. With about 2^{15} oracle queries for an initial setup and about 2^{15} queries for each block, the attacker can determine the first two bytes of plaintext in each block. The attack does require that the attacker also know the first two bytes of plaintext of any one block to bootstrap the process, but we show how to likely obtain these bytes in the majority of circumstances.

We believe that the presence of this attack will require future versions of OpenPGP software to disable the ad-hoc integrity check in the OpenPGP CFB mode.

2 Cipher-Feedback (CFB) Mode

This section will first describe the standard Cipher Feedback (CFB) mode of operation for block ciphers. The particular variant of CFB mode that is used in OpenPGP will then be described.

The standard CFB mode, by itself, does not appear to be affected by the results in this paper. However, if the data that has been encrypted using standard CFB mode has also been padded to produce an integer number of blocks of plaintext and there exists an oracle for determining when an encrypted message has been correctly padded, the techniques introduced in this paper along with the ideas in [13, 2] can be used to decrypt part or all of any ciphertext block.

2.1 Standard CFB Mode

We describe the standard CFB mode of operation as described in ANSI X9.52 [1] and NIST Special Publication 800-38A [7]. We will assume that the block size of the underlying block cipher, the block size of the CFB mode and the size of the feedback variable are all b bytes, since this is the case for the variant used by OpenPGP. We are doing this simply for ease of explanation and note that nothing in this paper depends upon this fact.

Let $E_K(\cdot)$ be encryption with the symmetric key K by the underlying block cipher. Let $\oplus$ be bitwise exclusive-or. The plaintext message to be encrypted will be $M = (M_1, M_2, \ldots, M_n)$ where each M_i is b bytes long. A random b-byte initialization vector IV is required in order to produce the ciphertext $C = (C_1, C_2, \ldots, C_n)$ as

$$\begin{aligned}
C_1 &= E_K(IV) \oplus M_1 \\
C_2 &= E_K(C_1) \oplus M_2 \\
&\vdots \\
C_i &= E_K(C_{i-1}) \oplus M_i \\
&\vdots \\
C_n &= E_K(C_{n-1}) \oplus M_n.
\end{aligned}$$

2.2 OpenPGP CFB Mode

The OpenPGP Message Format [4] uses a variant on the standard CFB mode. The main difference with the OpenPGP variant is that a plaintext initialization vector, as described above, is not used, but instead a random block R is encrypted as the first block of ciphertext. Two bytes of R are repeated in the second block in order to quickly check whether the session key K is incorrect upon decryption. We note that this "quick check" is really an integrity check on the key and it is this ad-hoc integrity mechanism, used in a mode of operation that wasn't designed to accommodate it, that allows the attack.

Let $\overline{0}$ be the b-byte all zero block. By $X_{i,j}$ or $[X]_{i,j}$ we will mean the ith and jth bytes of the block X and by X_{i-j} or $[X]_{i-j}$ we will mean the ith through jth bytes of X. The concatenation operation will be represented by $||$. Then, using notation as introduced in the previous section, the ciphertext $C = (C_1, C_2, \ldots, C_{n+2})$ is computed as

$$\begin{aligned}
C_1 &= E_K(\bar{0}) \oplus R \\
C_2 &= E_K(C_1)_{1,2} \oplus R_{b-1,b} \\
C_3 &= E_K([C_1]_{3-b}||C_2) \oplus M_1 \\
C_4 &= E_K(C_3) \oplus M_2 \\
&\vdots \\
C_i &= E_K(C_{i-1}) \oplus M_{i-2} \\
&\vdots \\
C_{n+2} &= E_K(C_{n+1}) \oplus M_n.
\end{aligned}$$

We note here that C_2 is not a full b-byte block, but instead consists of just 2 bytes. We will leave this slight abuse of notation as it will be useful later on.

The random number R is not sent to the receiver to use directly. Its purpose is to provide a random initial value on which to apply the block cipher and encrypt the first plaintext block M_1 and thus plays the role of the initialization vector in standard CFB mode. Note though that repetition of the two bytes of R in the computation of C_1 and C_2 allows the recipient to quickly check, after decrypting only blocks C_1 and C_2 whether or not the session key is likely correct. This is done by comparing the $b+1$st and $b+2$nd blocks with the $b-1$st and bth blocks. If they match, then the current key was likely the one used to encrypt the data and the remaining blocks can be decrypted. If they don't match, then the key is likely in error, and decryption can be halted. The OpenPGP specification (RFC 2440) describes this "quick check" and most implementations of OpenPGP include it. However, this is really an ad-hoc integrity check that leaks crucial information, as we shall see.

3 Attacking the OpenPGP CFB Mode

This section will describe the attack in detail. First we will describe the oracle required and the information that can be obtained from a successful oracle query. Then we will look at the format of the OpenPGP messages that are being decrypted and show that certain bits of M_1 can be predicted. We will then use the oracle and the known plaintext bits to determine 16 bits from any block of ciphertext.

3.1 The Oracle

This attack requires the presence of an oracle $\mathcal{O}$ that, when given a purported ciphertext C encrypted using the OpenPGP CFB mode of operation with a given key, will correctly determine whether or not the integrity check described in Section 2.2 was successful. We note that this oracle is likely to be implemented in practical implementations of OpenPGP. RFC 2440 requires that implementations implement this check to prevent obviously incorrect ciphertexts from being decrypted. Further details on the practical aspects of implementing this oracle will be discussed in Section 3.3.

Let's assume that such an oracle does, in fact, exist. Then if the oracle query is successful we know that for the purported ciphertext C,

$$[C_1]_{b-1,b} \oplus [E_K(\bar{0})]_{b-1,b} = C_2 \oplus E_K(C_1)_{1,2}. \tag{1}$$

We note that C_1 and C_2 are known since they are part of the ciphertext C. If we knew $E_K(C_1)_{1,2}$, then we could determine $[E_K(\bar{0})]_{b-1,b}$ and similarly if we knew $[E_K(\bar{0})]_{b-1,b}$, then we could determine $E_K(C_1)_{1,2}$. The method for the attack is now clear. We need to construct a message so that we know $E_K(C_1)_{1,2}$, that will allow us to obtain $[E_K(\bar{0})]_{b-1,b}$. This value is the same for all messages encrypted using K. Then, we can use that information to determine $E_K(C_1)_{1,2}$ in specially constructed messages, from which we will get the first two bytes of any plaintext block.

3.2 Obtaining Some Known Plaintext

In order for the attack described in this paper to work, the first two bytes of any one message block M_i must be known by the attacker. In this section, we will describe how an attacker can plausibly determine the first two bytes of M_1 in the majority of circumstances.

According to RFC 2440, the message M that is encrypted using the OpenPGP CFB mode consists entirely of a complete OpenPGP message. Both GnuPG (available at `http://www.gnupg.org`) and PGP Freeware (available at `http://www.pgp.com`) compress data by default before encrypting it. Thus, in the vast majority of circumstances, the encrypted message will consist of a compressed data packet. We will examine this situation first.

When the data being encrypted is a compressed data packet, the first two bytes of this packet are very predictable. The first byte consists of a "packet tag", which indicates the format of the packet contents. If the packet is compressed, then this packet tag will typically take the value `0xA3`. The second byte indicates the compression algorithm used. Typically this will be `0x01` to represent the ZIP compression algorithm. Other common values for the second byte are `0x00` for uncompressed data, `0x02` for ZLIB compression and `0x03` for BZip2 compression. Thus, if the attacker knows that the encrypted data is compressed, then the first two bytes will either be known or can be determined by trying a small number of possible values. If it is not known that the data is a compressed data packet, then the attacker can reasonably guess that it has been and thus guess the first two bytes.

If it is known that the encrypted data is not a compressed data packet, then there are a small number of choices for the first two bytes of M_1. The first byte will again be a "packet tag" to indicate the format of the packet contents. There are only a small number of possible packet tags likely to be used (14 are defined in the OpenPGP RFC). The second byte will either be the length of the encrypted message, or a portion of the length, depending upon whether or not more than one byte is required to represent the length. It is not unreasonable to assume that the attacker may know, or be able to guess, the packet type of the encrypted message and will likely be able to deduce the length from the message itself. Thus, it is not unreasonable to assume that the attacker will know, or be able to deduce the first two bytes of M_1.

Once the attacker knows $[M_1]_{1,2}$ then, from the definition of the OpenPGP CFB mode, it immediately also knows $E_K([C_1]_{3-b}||C_2)_{1,2}$. In the general case, if the attacker knows $[M_{i+1}]_{1,2}$ for $i \geq 1$ then it also knows $[E_K(C_i)]_{1,2}$. We will assume that the attacker knows one of these values.

3.3 The Initial Setup Work

In this section we will describe how the attacker can determine $[E_K(\overline{0})]_{b-1,b}$. We will assume that the attacker has intercepted $C = E_K(M)$ and knows the first two bytes of some plaintext block. As we saw in the last section, this is not an unreasonable assumption. We will first consider the situation where the attacker knows the first two bytes of M_1, next we will consider the situation where the attacker knows the first two bytes of M_{i+1} for $i \geq 1$.

When the attacker knows the first two bytes of M_1, the attacker will construct a ciphertext $C' = ([C_1]_{3-b}||C_2, D, C_3, C_4, \ldots)$ for particular values of D and submit it to the oracle to determine whether or not it is a properly constructed ciphertext. In other words, the oracle will determine whether or not the integrity check in the OpenPGP CFB mode was successful. When it is successful, we can use Equation (1) to determine $[E_K(\overline{0})]_{b-1,b}$.

In this situation, the attacker should use the following algorithm to determine $[E_K(\overline{0})]_{b-1,b}$.

1. Let D be a two byte integer representing the value 0.
2. Construct $C' = ([C_1]_{3-b}||C_2, D, C_3, C_4, \ldots)$.
3. Submit C' to the oracle $\mathcal{O}$. If the oracle returns "success" then

$$[E_K(\overline{0})]_{b-1,b} = C_2 \oplus D \oplus E_K([C_1]_{3-b}||C_2)_{1,2}.$$

Otherwise, let $D = D + 1$ and goto Step 1.

The correctness of this result follows immediately from Equation (1), the construction of C' and the fact that we are exhausting over all possible values of D. We note that from the previous section, the attacker knows $E_K([C_1]_{3-b}||C_2)_{1,2}$ and thus can, in fact compute $[E_K(\overline{0})]_{b-1,b}$.

We will now consider the more general case when the attacker knows the first two bytes of M_{i+1} for $i \geq 1$. This time the attacker will construct a ciphertext $C' = (C_i, D, C_3, C_4, \ldots)$ and proceed as in the previous case. The attacker would use the following algorithm to determine $[E_K(\overline{0})]_{b-1,b}$.

1. Let D be a two byte integer representing the value 0.
2. Construct $C' = (C_i, D, C_3, C_4, \ldots)$.
3. Submit C' to the oracle $\mathcal{O}$. If the oracle returns "success" then

$$[E_K(\overline{0})]_{b-1,b} = [C_i]_{b-1,b} \oplus D \oplus [E_K(C_i)]_{1,2}.$$

Otherwise, let $D = D + 1$ and goto Step 1.

Here we note that the attacker knows $[E_K(C_i)]_{1,2}$ from the results in the previous section and thus can also compute $[E_K(\overline{0})]_{b-1,b}$.

It is clear that in either of these situations the oracle will return "success" for some value of D less than 2^{16}. Thus, we would expect that, on average, our attacker would require about $2^{15} = 32,768$ oracle queries in order to determine $[E_K(\overline{0})]_{b-1,b}$.

3.4 Determining 16 Bits of Any Plaintext Block

Once our attacker has determined $[E_K(\overline{0})]_{b-1,b}$, the first two bytes of any plaintext block can be determined with about 2^{15} queries to the oracle. It is a simple variation on the algorithms in the previous section that provides it to the attacker.

In order to determine $[M_{i+1}]_{1,2}$ for any $i \geq 1$ the attacker should use the following algorithm.

1. Let D be a two byte integer representing the value 0.
2. Construct $C' = (C_{i+2}, D, C_3, C_4, \ldots)$.
3. Submit C' to the oracle $\mathcal{O}$. If the oracle returns "success" then

$$[E_K(C_{i+2})]_{1,2} = [C_{i+2}]_{b-1,b} \oplus D \oplus [E_K(\overline{0})]_{b-1,b}.$$

   Otherwise, let $D = D + 1$ and goto Step 1.
4. Then $[M_{i+1}]_{1,2} = [E_K(C_{i+2})]_{1,2} \oplus [C_{i+3}]_{1,2}$.

Again, the correctness of this result follows immediately from Equation (1), the construction of C' and the fact that we are exhausting over all possible values of D. As in the previous section we would expect that the attacker would require on average about 2^{15} oracle queries to determine the first two bytes of any plaintext block.

In the general case where the first two bytes of M_1 are not already known, they can be obtained by a simple modification to the above algorithm in which C_{i+2} is replaced by $[C_1]_{3-b}||C_2$.

3.5 The Attack Without Plaintext

We note that the attack can be implemented even if it is not possible to know the first two bytes of some plaintext block. In this situation, we can simply replace the assumed known value $[M_1]_{1,2}$ with an indeterminate, say Z and implement the algorithms in Sections 3.3 and 3.4. Note that now all of the formulae still carry through, including the "$\oplus Z$" term. Now instead of actually determining $[M_{i+1}]_{1,2}$ for any $i \geq 1$, we determine $[M_{i+1}]_{1,2} \oplus Z$. If enough of these values are recovered and if the values of the $[M_{i+1}]_{1,2}$ can be bounded, then Z can be determined, thus revealing the plaintext.

For example, if it is known that all of the M_i are ASCII text, then it wouldn't take very many values of $[M_{i+1}]_{1,2} \oplus Z$ to be recovered before Z could be determined.

Thus, we see that under the reasonable assumption that an attacker can determine the first two bytes of any one message block, with one-time initial work of about 2^{15} oracle queries, the first two bytes of any other message block can be determined with about 2^{15} oracle queries per message block. The following section will consider the question of the likelihood of this particular oracle existing.

3.6 Extending the Attack To Other Modes

We note that this attack is not really an attack on CFB mode encryption, but an attack on the two repeated bytes in the first two blocks of an OpenPGP encrypted message. It is likely that similar attacks would be possible with any non-authenticated encryption mode whenever the decryptor checks for repeated bytes. For example, Hal Finney has pointed out that a similar attack is possible against CBC mode if the decryptor checks for such a plaintext stutter [8].

As with the attack on padding in CBC mode, we note that in all of these situations the decryptor is checking for a specific redundancy when using a non-authenticated mode. This practice leaks too much information to the attacker. Such checks should be disabled or an authenticated mode of operation should be used whenever possible.

4 The Attack in Practice

In previous sections we showed that if a certain oracle exists, then under reasonable assumptions it is possible for an attacker to determine the first two bytes of any plaintext block. This section will examine how likely it is that the required oracle will exist in practice.

We first note that the required oracle $\mathcal{O}$ simply implements the integrity check required in the OpenPGP standard. Thus it is not unreasonable to expect that most implementations would leak this kind of information. This is not a very "powerful" oracle in the sense that it is not leaking a great deal of information. We contrast this with the oracle required in the attack on the OpenPGP CFB mode described in [12, 10]. In that attack only a single oracle query is required to determine the entire plaintext, however, the oracle must return the decryption of the chosen ciphertext. In most environments it is not likely that the attacker will actually have access to the decryption of the chosen ciphertext. It is not unreasonable though to assume that error information is obtained either directly, or through side-channels.

4.1 Non-Server-Based OpenPGP Users

By a "non-server-based OpenPGP user" we refer to a human user interacting with an OpenPGP-enabled application. This is, by far, the most common scenario of OpenPGP-based applications. In this scenario it is not unreasonable to assume that some error information regarding the decryption of any ciphertext will be leaked to an attacker. However, it is not likely at all that a human user will attempt to decrypt over 32,000 messages whose decryptions actually fail without realizing that there is a problem and discontinuing.

Thus, we view an attack in this situation as unlikely and will not consider it any further.

4.2 Server-Based OpenPGP Users

A "server-based OpenPGP user" is an automated process that responds to requests that have been encrypted for it using OpenPGP. It attempts to decrypt the request and respond to it appropriately. This is a much less likely scenario than in the previous section. In this situation it is more likely that information on errors (including decryption errors) will be returned to the requester, which in this case could be an attacker.

There are at least two ways in which an attacker could gain information that would instantiate the oracle. The server could return an error directly to the attacker that the integrity check in the OpenPGP CFB mode failed. As we will see in the next section, some common OpenPGP toolkits do, in fact return error codes which could allow server-based OpenPGP users to, unwittingly, instantiate the oracle. Even if this error code is not returned, information on whether or not the integrity check was successful can likely be obtained through timing or other information. RFC 2440 says that the integrity check "allows the receiver to immediately check whether the session key is incorrect". Thus, most implementations would abandon the decryption process immediately if the check failed thereby allowing timing attacks to determine the result of the check. As we will see in the next section, this is what happens.

We also need the oracle to use the same symmetric key K each time that it is invoked. This is not difficult to do. After constructing the ciphertext C' as described in previous sections, this ciphertext should simply be packaged in a standard OpenPGP message. The Public-Key Encrypted Session Key Packet in this message should always contain the same value as in the original OpenPGP message that is being attacked. This packet will contain the key K encrypted for the victim. Each time that the chosen ciphertext is sent to the victim, he/she will decrypt and thus use the same key K each time.

4.3 Common Toolkits May Instantiate the Oracle

To determine the likelihood of this oracle being instantiated we looked at two common toolkits that implement the OpenPGP RFC. We considered GnuPG 1.2.6 [9] and Cryptix OpenPGP [6].

In GnuPG 1.2.6, the integrity check is performed in the `decrypt_data()` function call. If the integrity check is not successful, then the error `G10ERR_BAD_KEY` is returned and decryption is abandoned. Thus, it is not unreasonable to expect that some server-based applications based upon this toolkit would leak this error information either directly, or based upon timing information.

In the Cryptix OpenPGP toolkit, the `PGPEncryptedDataPacket.decrypt()` method performs the integrity check. If the integrity check is not successful then an exception is thrown with "No good session key found" and decryption doesn't proceed. Thus, again, it is not unreasonable to expect that some server-based applications based upon this toolkit would leak this error information either directly, or based upon timing or other information.

4.4 Implementing the Attack

We implemented the attack on GnuPG 1.2.4. As it turns out, GnuPG is very helpful in that it appears to display the error "`decryption failed: bad key`" if and only if our oracle does not return success.

We encrypted data with compression turned off and without MDC (see next Section). This was only for ease of implementation, as we have seen (and shall see) this is not required. We then implemented the algorithms in Sections 3.3 and 3.4 as batch scripts cycling through all possible values of D. When we did not get a "`decryption failed: bad key`" we knew that the integrity check was successful and could utilize the given formulae to produce the plaintext. We note that in all of our experiments we only received one value of D that did not give a "`decryption failed: bad key`" error.

Implemented on a 1.8 GHz Pentium M processor running Windows XP Professional, it took under 2 hours to exhaust all values of D. Thus, with less than 4 hours of work an attacker could obtain the first two bytes of any plaintext block. The first two bytes of additional plaintext blocks could be obtained with an additional 2 hours each.

4.5 The Effect of Compression

When 64-bit blocks are used an attacker can obtain at most 25% of the plaintext and when 128-bit blocks are being used at most 12.5%. If the plaintext is uncompressed data this would be devistating. However, typically plaintext OpenPGP data is compressed. In this situation it is not clear if obtaining even 25% of the compressed data will compromise any of the uncompressed data. This is a big mitigating factor against the attack in practice.

5 Attack Prevention

In this section we examine two potential methods for avoiding this attack. One method does not work, the other does.

5.1 Integrity Protected Data Packet Doesn't Work

The OpenPGP RFC is currently up for revision [5] and a new packet type will be included that provides additional integrity protection for encrypted data. GnuPG also implements this additional packet type, called the Symmetrically Encrypted Integrity Protected Data Packet. Encryption using this packet type differs from the non-integrity protected version in two ways. First, the OpenPGP CFB mode of encryption as described is not used. Instead $b + 2$ random bytes, with the $b + 1$st and $b + 2$nd bytes matching the $b - 1$st and bth bytes, are prefixed to the message and then the standard CFB mode is used to encrypt the prefixed message. As previously, the repeated bytes should be checked and decryption abandoned if they do not match. Second, the entire plaintext message, including the prefix is hashed. The hash value, known as an MDC, is appended to the message and encrypted using CFB mode as well.

Let us first consider the modified CFB mode of operation. We note that, in general, the attack described still works with slight modifications (e.g. replace C_2 with $[C_2]_{1,2}$). However, it will likely now become more difficult for an attacker to obtain the first two bytes of a plaintext message block in order to bootstrap the attack. Notice that now the suggested known plaintext will be in bytes 3 and 4 of the plaintext corresponding to C_2. If the first two bytes of any plaintext message block is known, however, the attack will still be valid.

The purpose of the hash is to detect modifications of the message by the attacker. The attack described in this paper involves modifications to the message and the hash will, in fact, detect it. However, since the check of the hash occurs **after** the decryption of the entire plaintext and the ad-hoc integrity check of the bytes in C_1 and C_2 occurs **before** the decryption of the entire plaintext, it is still likely that information that instantiates the oracle will be leaked. In fact, since a hash will now need to be computed before determining whether or not the plaintext is valid in addition to decrypting the message, it is likely that timing attacks to determine the information will be more effective. We note that GnuPG implements this new packet type and still returns different error codes for the two errors and abandons decryption if the repeated bytes do not match.

Thus, this packet type, by itself, will not prevent this attack, although it may make it more difficult to start.

5.2 The Solution

The only method that appears to always work in thwarting this attack is to not instantiate the required oracle. Thus, implementations should not do the check that the repeated bytes in the first two blocks match. If the non-integrity protected packet type is being used, then the data should all be decrypted and an attempt should be made at parsing it. If the integrity protected packet type is being used, then the entire ciphertext should, again, be decrypted and the hash calculated and checked. If it doesn't match, then an error can be thrown.

Unfortunately, for backwards compatibility with the substantial installed user-base it is not possible to remove these random repeated bytes from the encrypted data format. However, future versions should simply ignore these bytes.

If a similar "quick check" that would allow OpenPGP users to quickly determine whether or not the given symmetric key is correct is required, then one possible solution is to include a cryptographic hash of the symmetric key with the ciphertext. The message recipient could then compute the hash of the purported symmetric key and compare it with the given value before decrypting. Note that this solution would not provide an integrity check on the entire message and would require changes to the OpenPGP RFC.

6 Conclusion

We have described an attack on the OpenPGP CFB mode of operation. This attack works under reasonable assumptions about the knowledge of certain plaintext bytes and requires an oracle which is likely instantiated in most applications using OpenPGP. However, since the attack requires 2^{15} oracle queries, on average, for the initial setup and 2^{15} oracle queries to determine the first two bytes of any plaintext block, it likely won't effect applications with human end users. Server-based applications would be more vulnerable to attack though. In order to thwart this attack, future implementations should not perform the ad-hoc integrity check in the OpenPGP CFB mode.

Acknowledgements. We would like to thank Jon Callas, Hal Finney and Don Johnson for their helpful comments.

References

1. ANSI X9.52 – 1998, "Triple Data Encryption Algorithm Modes Of Operation", American National Standards Institute, July 29, 1998.
2. J. Black and H. Urtubia, "Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption," In *Proceedings of the 11th USENIX Security Symposium*, pp. 327-338, 2002. Available at `http://www.usenix.org/events/sec02/full_papers/black/black_html/`
3. D. Bleichenbacher. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1." In H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pp. 1 - 12. Springer Verlag, 1998.
4. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP Message Format," RFC 2440, Nov 1998.
5. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP Message Format," draft-ietf-openpgp-rfc2440bis-XX.txt, *work in progress.*
6. Cryptix OpenPGP, 20041006 snapshot. Available at `http://www.cryptix.org/`
7. M. Dworkin, "Recommendation for Block Cipher Modes of Operation," US Department of Commerce, *NIST Special Publication 800-38A*, 2001. Available at `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`
8. H. Finney, *personal communications.*
9. The GNU Privacy Guard, version 1.2.6. Available at `http://www.gnupg.org/`
10. K. Jallad, J. Katz, and B. Schneier, "Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG," In *Proceedings of the 5th International Conference on Information Security*, pp. 90-101, 2002.
11. B. Kaliski, "PKCS #7: RSA Encryption, Version 1.5," RFC 2313, Mar 1998.
12. J. Katz and B. Schneier, "A Chosen Ciphertext Attack against Several E-Mail Encryption Protocols," In*Proceedings of the 9th USENIX Security Symposium* pp. 241-246, 2000. Available at `http://www.usenix.org/publications/library/proceedings/sec2000/katz.html`
13. S. Vaudenay, "Security Flaws Induced by CBC Padding-Applications to SSL, IPSEC, WTLS . . .," In Lars Knudsen, editor *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pp. 534-545, Springer-Verlag, 2002.